Design of DCache Storage Resource Manager Implementation

Table of Contents

Purpose of this document

This document is intended to be read by the participants of the Design of dCache SRM review. The goal of the review is to recommend how to improve the SRM implementation so that it can satisfy the requirements of the LHC Experiments, OSG and WLCG users of the dCache Storage System with concentration on how to increase the performance and scalability of the product. The following requirement that needs to be satisfied by dCache SRM was received from CMS and was later confirmed by Atlas: “All SRM functions should scale to the limits of the hardware deployed, it should never be the limiting factor. “ The goal of the review is to recommend the implementation of the SRM component of the dCache and not the rest of dCache.

The purpose of this document is to provide the readers with all the information necessary to conduct the review. This documents describes SRM interface, technologies used to implement SRM, basic architecture of dCache and role of various components of dCache and how they are utilized by SRM. This documents provides references to relevant documents and to dcache code repository from where all dCache source code including code for dCache SRM can be obtained. The language of this document is not expected to be of the conference level quality, its main purpose is to inform and not to impress.

Introduction

Storage Resource Managers (SRMs) are middleware components whose function is to provide dynamic space allocation and file management and operations on shared storage components on the Grid. To learn about the concepts and ideas of GRID, please review the Anatomy and Physiology of the GRID in these two papers: [16],[17]. SRMs support protocol negotiation and a reliable replication mechanism. The SRM specification standardizes the interface, thus allowing for a uniform access to heterogeneous storage elements. The SRM standard allows independent institutions to implement their own

SRMs. SRMs leave the policy decision to be made independently by each implementation at each site. Resource Reservations made through SRMs have limited lifetimes and allow for automatic collection of unused resources thus preventing clogging of storage systems with "forgotten" files.

The storage systems can be classified on basis of their longevity and persistence of the data they store. Data can be considered to be temporary and permanent. For example disc caches might allow for spontaneous deletion of the files, while deletion of the file stored in a robotic tape storage can be very problematic. To support these notions, SRM defines three types of files and spaces: Volatile, Durable and Permanent. Volatile files can be removed by the system to make space for new files upon the expiration of its lifetime. Permanent files are expected to exist in the storage system for the lifetime of the storage system, unless explicitly deleted by the user. Finally Durable files have a both the lifetime associated with them and a mechanism of notification of owners and administrators of lifetime expiration but can not be deleted automatically by the system and require explicit removal.

SRM interface consists of the five categories of functions: Space Management, Data Transfer, Request Status, Directory and Permission Functions. Among the functions worth mentioning are srmReserveSpace which creates advanced space reservation with a user specified lifetime, and identified by a unique space token. The space token can later be utilized for storing files via srmPrepareToPut, srmPrepareToGet and srmCopy functions. For example srmPrepareToPut will take the list of files, file sizes, list client supported transfer protocols and space tokens. SRM interface utilizes Grid Security Infrastructure (GSI) for authentications. SRM service is a Web Service implementation of a published WSDL document. Fermilab SRM is based on and is an integral part of the dCache Distributed Disk Cache coupled with Enstore Tape Storage System (SRM or SRM code do not rely or depend on Enstore).

Source code, binaries and documentation access

dCache documentation and binaries are available at dcache web site at [9] . dCache is an open source project, description on how to gain anonymous read only access to subversion repository or to browse the repository using web interface is at [10]. Javadocs of generic SRM code can be found at [14].

Performance Profiling results

On January 26 I performed the profiling of the SRM code running in US CMS T1 production system running under relatively light load at the moment of testing. The results are published at [13].

Description

Technology and architecture overview

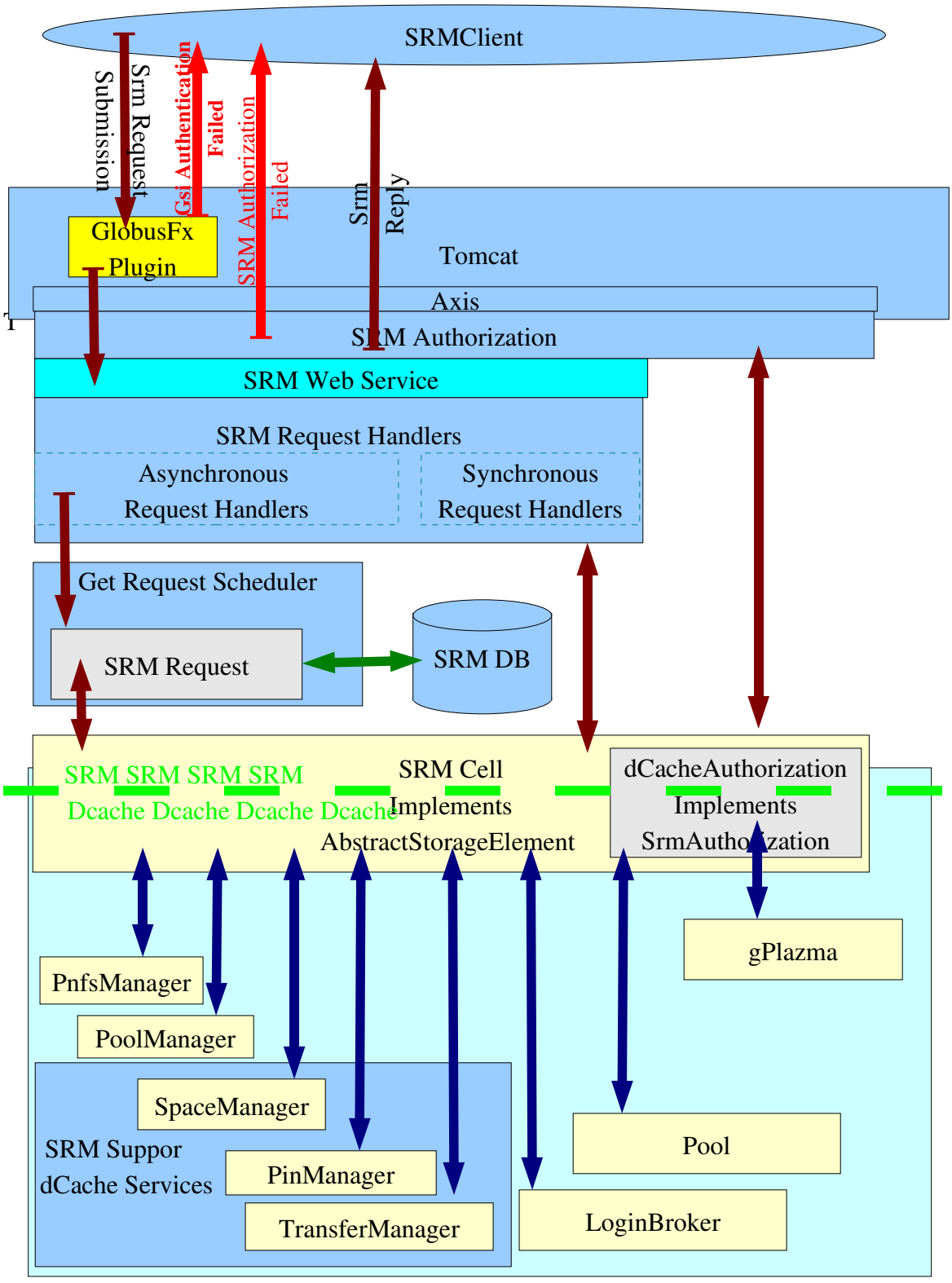
dCache SRM as well as the rest of the dCache components is a java program , usually executed with the Java Virtual Machine implementation by Sun Microsystem on Linux OS.

dCache SRM is Web Service implementing two standards defined by Storage Resource Manager V1.1 and Storage Resource Manager V2.2 specifications [1] ,[2]. SRM Web Services run on top of httpg (https using GSI instead of TSL sockets). GSI which stands for Grid Security Infrastructure is described in [3]. GSI protocol supports X509 based authentication and delegation of X509 based credential. dCache implementation uses Apache Tomcat [4] coupled with Apache Axis [5] as Web Service container containing dCache SRM as its application. dCache SRM uses Globus project plug-ins for Tomcat and Axis to make them aware of GSI Transport [8] .

dCache distributed application is build on top of Cells messaging system [6],[7] . Each instance of the java process which is a part of dCache application is a “Domain” running one or more “cells”. Cells can communicate with each other using messages. Message can be send to its destination with the sender waiting for a reply in a synchronous or asynchronous manner, or message can be send without sender waiting for a reply (one way communication). Once cells are deployed and declared “well known”, than the message sender does not need to concern itself with weather the message recipient is in the same Domain, on the same host or in Domain running on a different host.

From the point of view of the dCache application, its SRM is a *cell* that communicates with other components of dCache using cell messages. dCache SRM communicates with and/or relies upon the following services (cells) in dCache distributed application: gPlazma (authentication service), PnfsManager (Namespace and File Metadata service),

Login Broker(transfer endpoint registry), Pin Manager, Space Manager, PoolManager (client transfer request and HSM store/restore operations scheduler), Pool (dCache Managed Disk Data Storage). Illustrating the above description, the diagram bellow provides a high level architectural diagram of dCache SRM.



SRM Architectural Diagram.

Generic vs. dCache specific SRM Code

dCache SRM is implemented as a generic product that communicates with underlying storage though a well defined set of interfaces. Should it become necessary to implement an SRM interface to a different Storage System, all of the general SRM code could be reused without modifications, and only implementations of the SRM-to-Storage interfaces would be needed. All of the Generic SRM code is in the dCache module called "srm". There is no dependency in SRM code on any dCache spec code. All of the dCache specific code is located in dCache module called "dCache" with majority of the classes located in java package diskCacheV111.srm.dcache.

Dcache Configuration

dCache SRM has a large number of the configurable parameters, that set during the initializations with some of them modifiable during the runtime. All of these parameters are properties of the org.dcache.srm.util.Configuration class. This class also knows how to write itself into and read itself from an xml file. But this functionality is not used in dCache.

SRM-to-Storage interfaces

The interfaces are located in org.dcache.srm package.

org.dcache.srm.SRMAuthorization – abstraction of the storage specific authorization mechanism.
org.dcache.srm.SRMUser is interface is an instance of the authorization record or an object that captures authorization decision by storage authorization mechanism
org.dcache.srm.AbstractStorageElement - main interface between SRM and underlying storage.

The following callback interfaces are passed as arguments in AbstractStorageElement functions. They are used for asynchronous notifications of completion of execution of the functions.

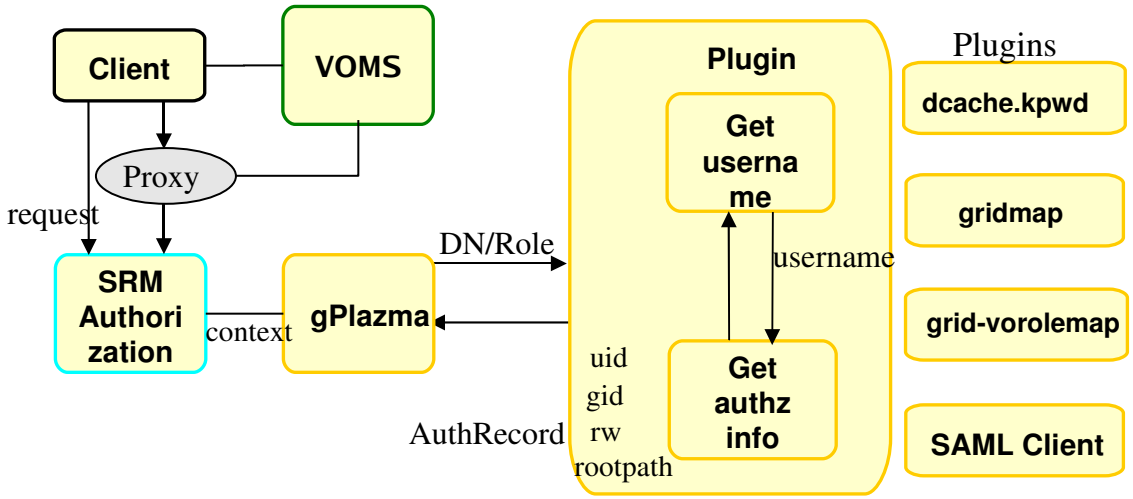
org.dcache.srm.ReserveSpaceCallbacks
org.dcache.srm.AdvisoryDeleteCallbacks

org.dcache.srm.PinCallbacks
org.dcache.srm.UnpinCallbacks
org.dcache.srm.PrepareToPutCallbacks
org.dcache.srm.CopyCallbacks
org.dcache.srm.PrepareToPutInSpaceCallbacks
org.dcache.srm.GetFileInfoCallbacks
org.dcache.srm.ReleaseSpaceCallbacks

Authorization of the users and operations

Storage specific implementation of the SRMAuthorization interface is passed the user credential and it gives back a Storage Specific SRMUser instance. The only two properties that SRM requiters is id and priority. Id has to be unique to the user and a priority is a number that is higher for the higher priority users. This number is than used by the SRM scheduler, when executing SRM Requests. Almost all methods of the AbstractStorageElement take SRMUser as an argument, and the Storage Specific implementation is responsible for interpreting the SRMUser and determining if the user has permission to perform a particular operation. Another interface that Storage Implementation provides is the org.dcache.srm.SRMUserPersistenceManager. This object knows how to store the SRMUser Authorization record in a persistent storage and how to retrieve the object using its id.

Diagram Bellow illustrates the authorization and authentication of the users by dCache implementation of SRM Authorization and its interaction with gPlazma (main dCache authorization mechanism).



Caching of User's credentials and Authorization decisions

In order to optimize the performance and reduce usage of memory SRM always only stores one copy of the user's credential (x509 cert) in memory. If the user performs more than one operation simultaneously, SRM will detect that it already has the user's credential and will make the following request to point to the same credential. If the credential that is supplied with the following request has a longer lifetime, SRM will substitute the credential stored with the new one for all requests by the same user. In order to support the execution of the requests after the restart, and authentication information to monitoring interface, SRM provides a persistent storage of the user's credential. This is a combination of the database record and a local file.

In order to minimize the load on the Authorization service and improve the performance, dCache implementation of the SRMAuthorization interface caches the AuthorizationRecords for short (3 minutes by default) period of time, so if multiple requests are executed simultaneously or in quick succession for the same user, irregardless of the user requests rate gPlazma will be contacted only once every 3 minutes. Again, to support the execution of the requests after the restart and to provide the authorization information to monitoring interface, SRM provides a database based persistent storage for AuthorizationRecords, based on JPA (Java Persistence API).

SRM Operations

Common logic executed for all SRM Operations.

Once the client connects to the Tomcat server hosting SRM application, using TCP/IP protocol, tomcat uses a org.globus.tomcat.coyote.net.HTTPSConnector from Globus Cog FX kit to create an instance of the class that will handle the connection. HTTPSConnector uses org.globus.tomcat.catalina.net.BaseHTTPSServerSocketFactory , which creates a wrapper around the TCP Socket which performs GSI version of SSL Handshake, During this handshake the authentication protocol is negotiated and credentials of the client and server are exchanged. All of this is handled by GSI authentication code from the globus library. The special instance of the Tomcat Http request Valve (a class that performs custom preprocessing of the http request before it is passed to the tomcat hosted application or web server) extracts the GSI Credential from the connected Gsi socket and puts it in the Http Request Context.

Later the Http request is passed to Axis application in Tomcat, which interprets the Soap message and and invokes one of the SRM Server Web Service implementation methods with correctly filled arguments. Once these methods return, the return values are converted to the Soap reply messages, which are embedded in the http reply, and axis returns control back to Tomcat.

The classes that contain the srm methods called by Axis are org.dcache.srm.server.SRMServerV1 and org.dcache.srm.server.SRMServerV2. All of the public methods before executing the logic specific to each of the operations, perform the following common tasks:

1. get user credential from the HttpRequestContext, that was previously put there by the tomcat valve.
2. perform authorization using Storage Specific implementation of SrmAuthorization interface
3. perform the requested action, if authorization is granted.
4. return the results.

In case of SRMServerV2, which implements all SRM V2.2 methods, the logic described above is encapsulated in the private method “handleRequest”, which uses reflection to find a handler class for each specific operation of srm v2.2 in org.dcache.srm.handler package, so that the name of the handler class is exactly the name of the srm operation invoked, but with first latter of the name capitalized. So if srm client invokes srmRm method, the methods of the org.dcache.srm.handler.SrmRm class will be invoked to satisfy this request.

Types of SRM Operations

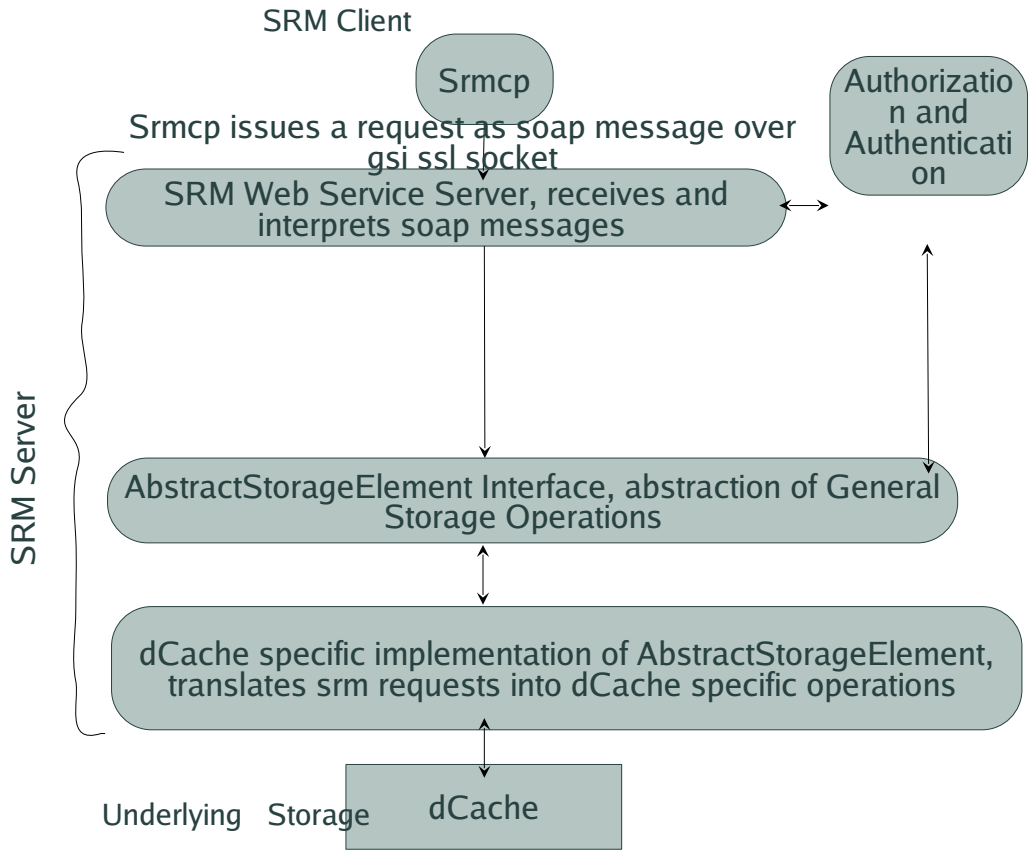
From the point of view of SRM Server there are two types of the SRM operations. Operations of the first type perform certain function on the server in one step web service request/response communication and server returns the results of such operation without changing the state of the SRM Server itself, while possibly making useful changes in of dCache as a whole. Examples of such operations are a listing of a directory and a removal of a file. We call these types of operations “blocking”.

Operations of the second type are performed by making a series of web service invocations and lead to a creation of the session objects on the server side identified by

the unique tokens. The server may continue to perform the requested operation initiated by the first invocation, and the client may periodically poll the status of the operation supplying the token as one of the arguments of the status Web Service Function. Example of one of such operations is srmPrepareToPut. SRM Client issues an PrepareToPut command for a file. SRM server returns the reply that contains the Status of the put operations which is initially “Queued” and a “Request Token”, srm client starts periodically poll the status of the “PrepareToPut” operation by issuing getStatusOfPrepareToPut command, supplying file name and token as its arguments. It gets back the Status object, that continues to have “Queued” Status, but eventually becomes “SrmSpaceAvailable”, and the TURL (Transfer URL) becomes available too. After that SRM Client performs the transfer of local file into the server identified by TURL . Than client notifies the SRM server that the transfer is complete by issuing the PutDone command that signals the server that it can destroy the server side session object and perform necessary cleanup and release of the temporary reserved resources. We call this sort of operations “non-blocking”.

Implementation of the blocking operations in dCache SRM

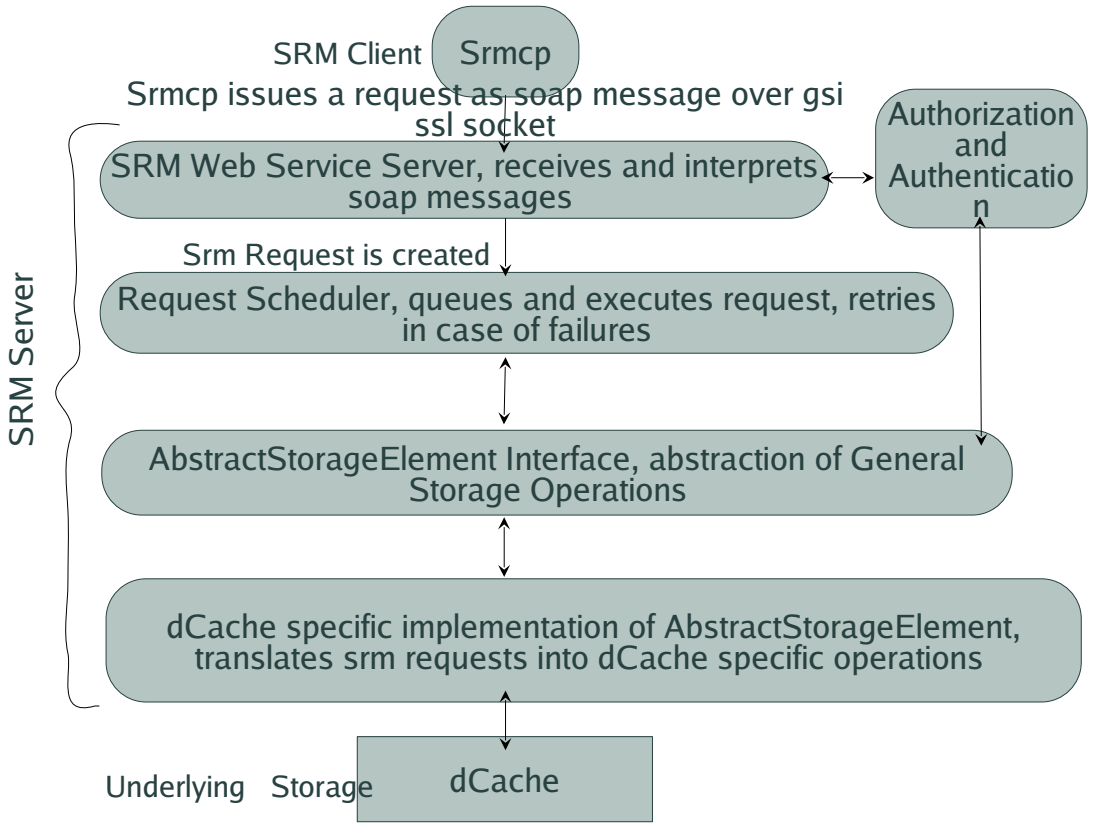
The implementations of the blocking operations are easy to understand. No threaded programming or event notifications or callbacks are used. The code is the simple sequence of actions that leads to either accomplishment of the requested action or a (partial) failure, followed by the generation and return of the structured reply object or a simple return. It is illustrated by the diagram on Picture 1..



Picture 1.

Implementation of the non-blocking operations in dCache SRM

Implementation of the non-blocking srm operations is much more complex, and is illustrated by Picture 2..



Picture 2.

The initial call leads to the creation of the Srm Request, of a particular type. The Request is a subclass of the org.dcache.srm.request.Request class, which is in its turn a subclass of org.dcache.srm.scheduler.Job.

Once Job is created, it can be passed to the instance of the Scheduler object that will execute the run method of the Job, using Scheduler's pool of threads. Execution of the run method of Job leads to its transition from one state to another. This is repeated until the

request reaches one of its final states. Some requests can contain a collection of the FileRequests. Each of the FileRequest is a Job as well. FileReqeusts are executed by a Scheduler as well. After initial request, clients starts to poll the status of the request using the request token and optionally SURL(s) of (a) file(s) in the request. This is translated in the invocation of the getRequestStatus methods of the concrete Request and FileRequest instances. Client can change the state of the request, file request or cancel the execution of the requests by issuing various srm commands (srmPutDone or srmAbortRequest for example) . This again leads to the execution of the methods of Request or FileRequests.

The concrete types of the requests and corresponding file requests, all of which are in the org.dcache.srm.request package are:

BringOnlineRequest and BringOnlineFileRequest

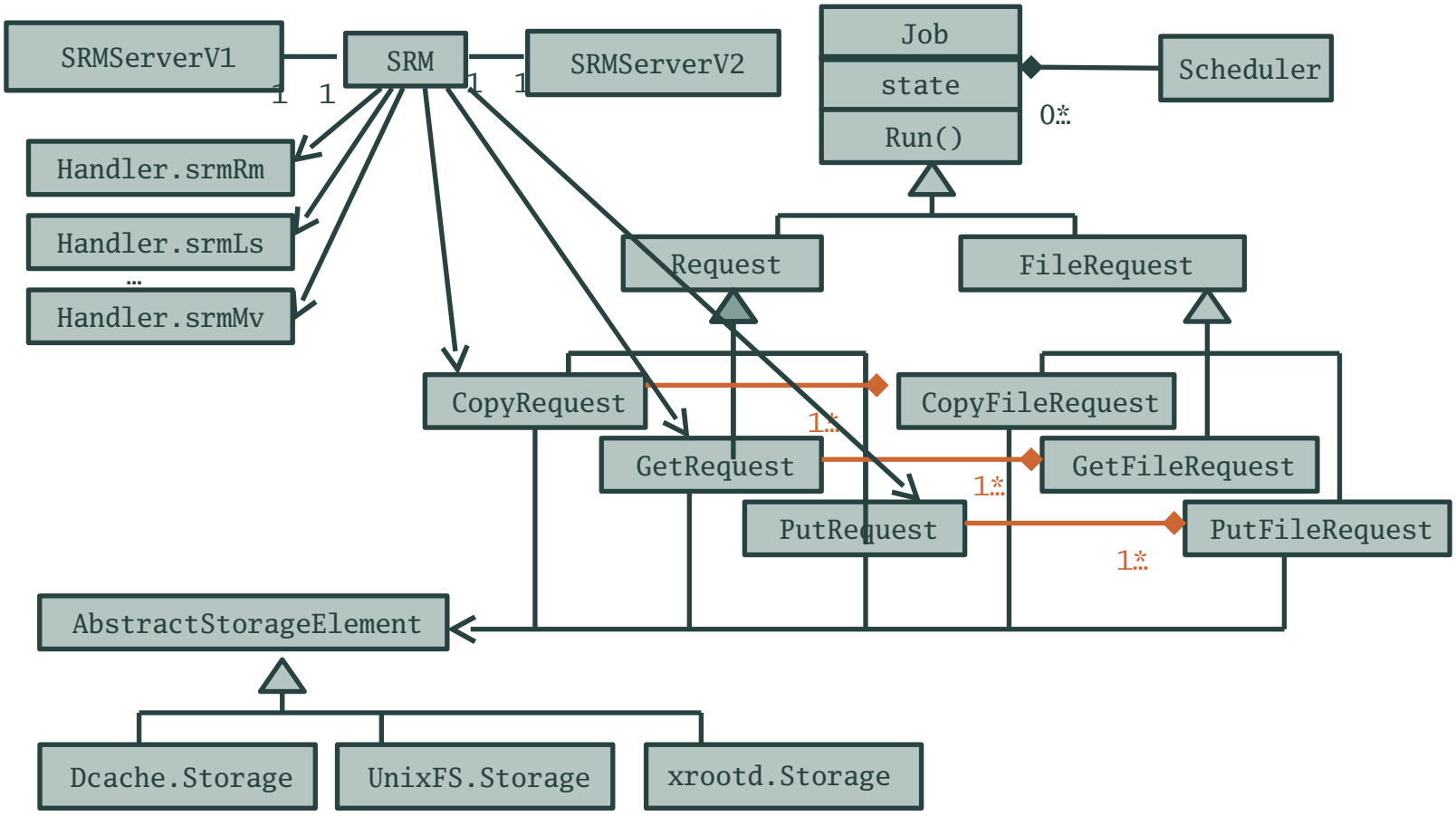
CopyRequest and CopyFileRequest

GetRequest and GetFileRequest

PutRequest and PutFileRequest

ReserveSpaceRequest has no corresponding file requests.

A simplified class diagram describing the relationships between these classes is on Picture 3. SRM Class Diagram



Picture 3. SRM Class Diagram

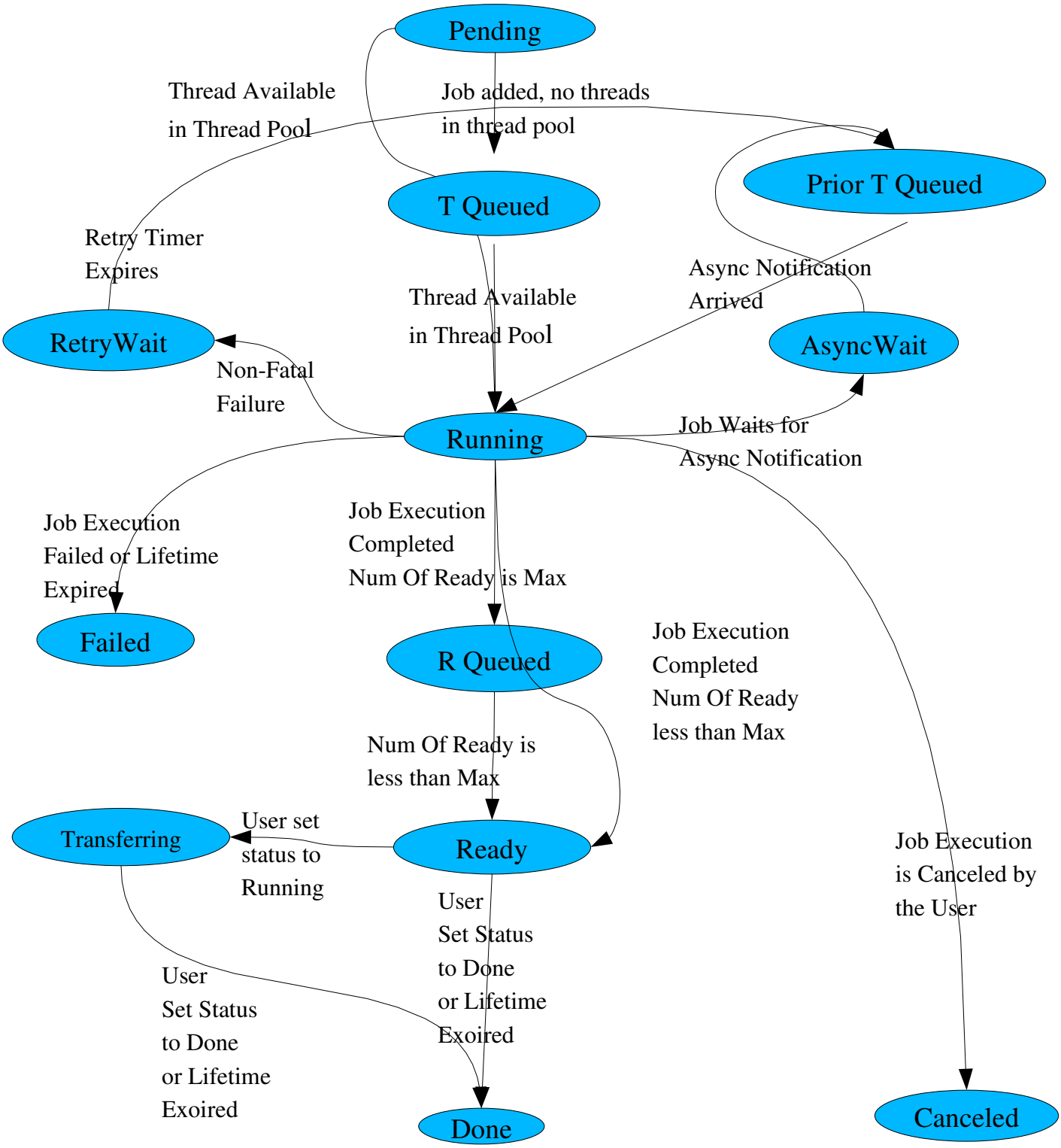
SRM Scheduler and Jobs State transitions.

The states that SRM Job can assume are described by the instances of the org.dcache.srm.scheduler.State class. Which of the state transitions are legal and illegal can be deduced from the logic of Job.setState() method.

The initial state is Pending and the final State is one of the three: Canceled, Done or Failed.

A somewhat incomplete diagram on Picture 4. SRM Scheduler Job's states and some of the state transitions. gives an impression about the possible state transitions for SRM Request and FileRequest. Execution of some of the instances of Jobs, depending on their type can only assume a subset of these states.

SRM Scheduler is the class that keeps a pool of threads that are used for execution of the instances of SRM Jobs. A detailed description of the Scheduler can be found in [11].



Picture 4. SRM Scheduler Job's states and some of the state transitions.

Persistence of the Srm Requests in database.

SrmRequests are persisted in a database. This allows SRM to server to continue execution of the non-blocking SRM requests after restart. Another function of the database storage for the request is data source for monitoring application SrmWatch, which provides several web views on the current state of the SRM system as well as the historic information about system. Using various views you could see details of an individual srm request, its status and history of execution, you could see all the requests submitted in a given time period as well as various statistical graphs. The query page is a great search tool and results of a query can be displayed as both table and time diagram.

SRM in default configuration keeps the requests objects in memory as long as it has memory to do so. Hence during normal operation it does not notice any updates to the database. This is great for performance (no need to wait for the database to return the data) but means that multiple SRM instances cannot use the database for communicating with each other. This could be easily solved by making sure that each instance of SRM does keep in memory only the requests it controls and updates. Another problem to horizontal scalability is updates to existing requests that are sent to the srm that does not control a particular request, but this could be handled by implementing a pier-2-pier communication layer between the SRM instances that would forward such update requests to the correct instance of SRM server within the same dCache.

I order to persist a request in a database we store the following objects in their corresponding sql tables: User´s credential, User Authorization Record, Request, File Request, and a set of dated State Transitions. The interfaces to the persistent storage are

org.dcache.srm.scheduler.JobStorage

org.dcache.srm.request.RequestCredentialStorage

org.dcache.srm.request.FileRequestStorage

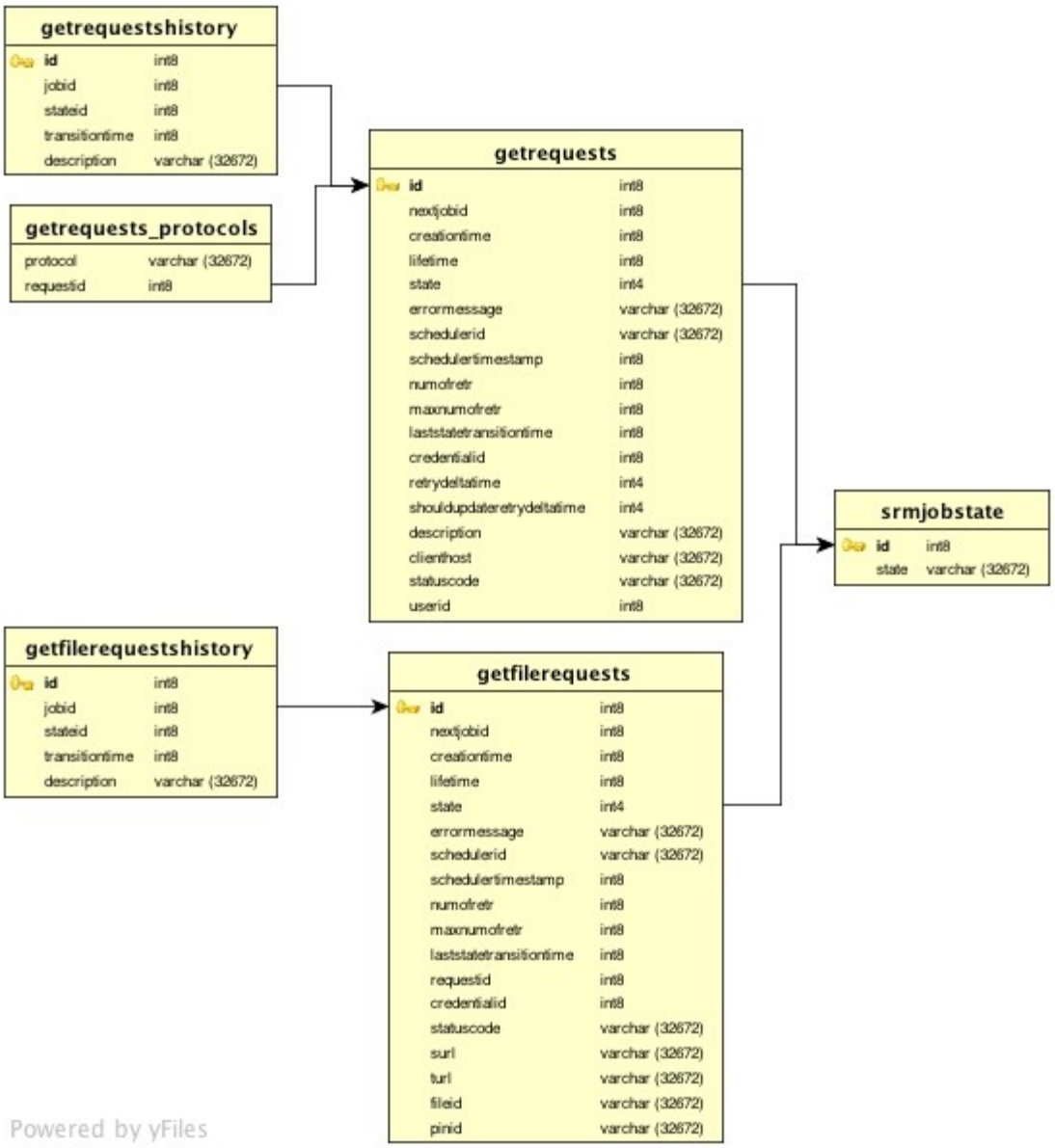
org.dcache.srm.request.RequestStorage

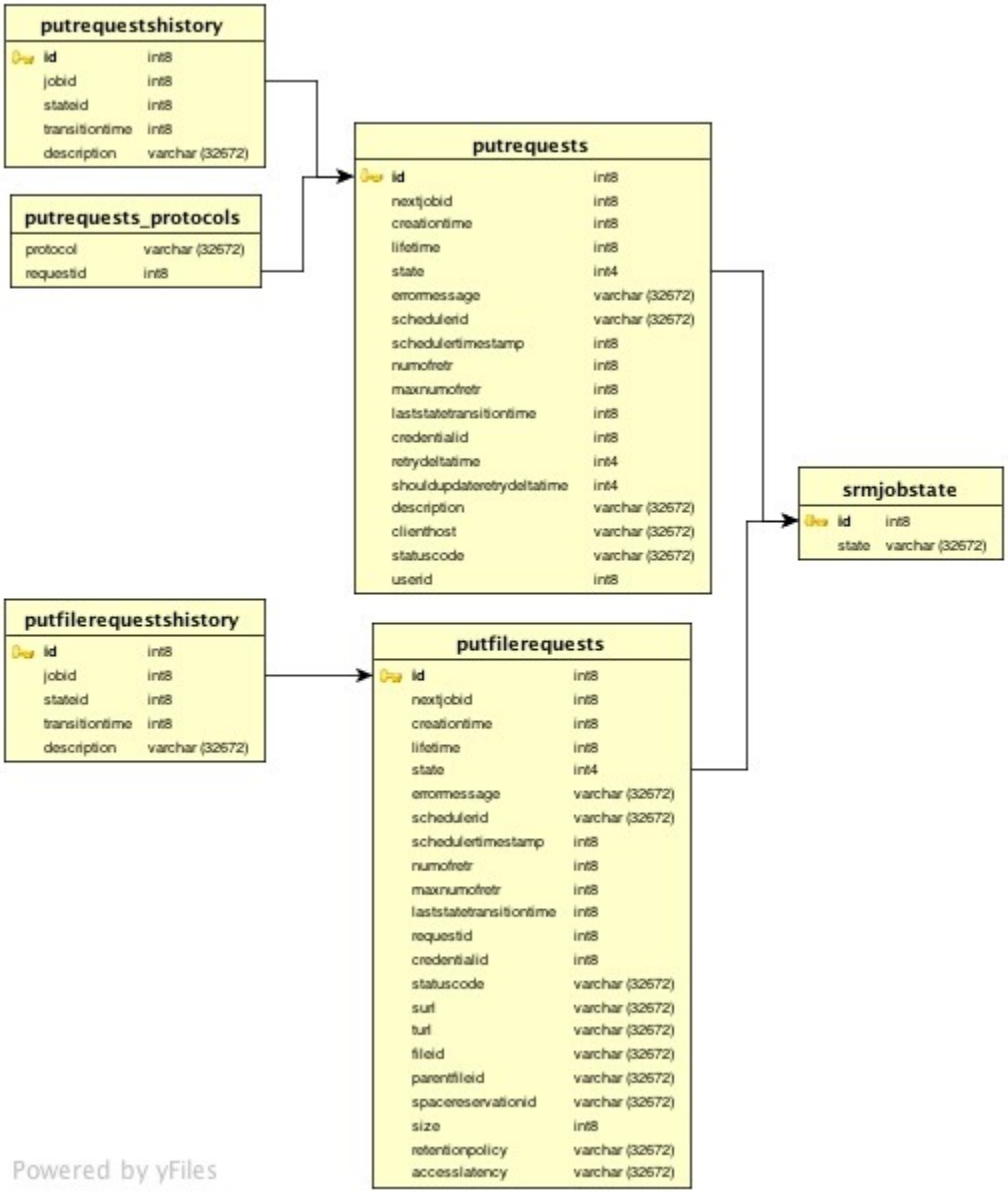
and their sub interfaces for specific types of the requests and file requests (all are in package org.dcache.srm.request.

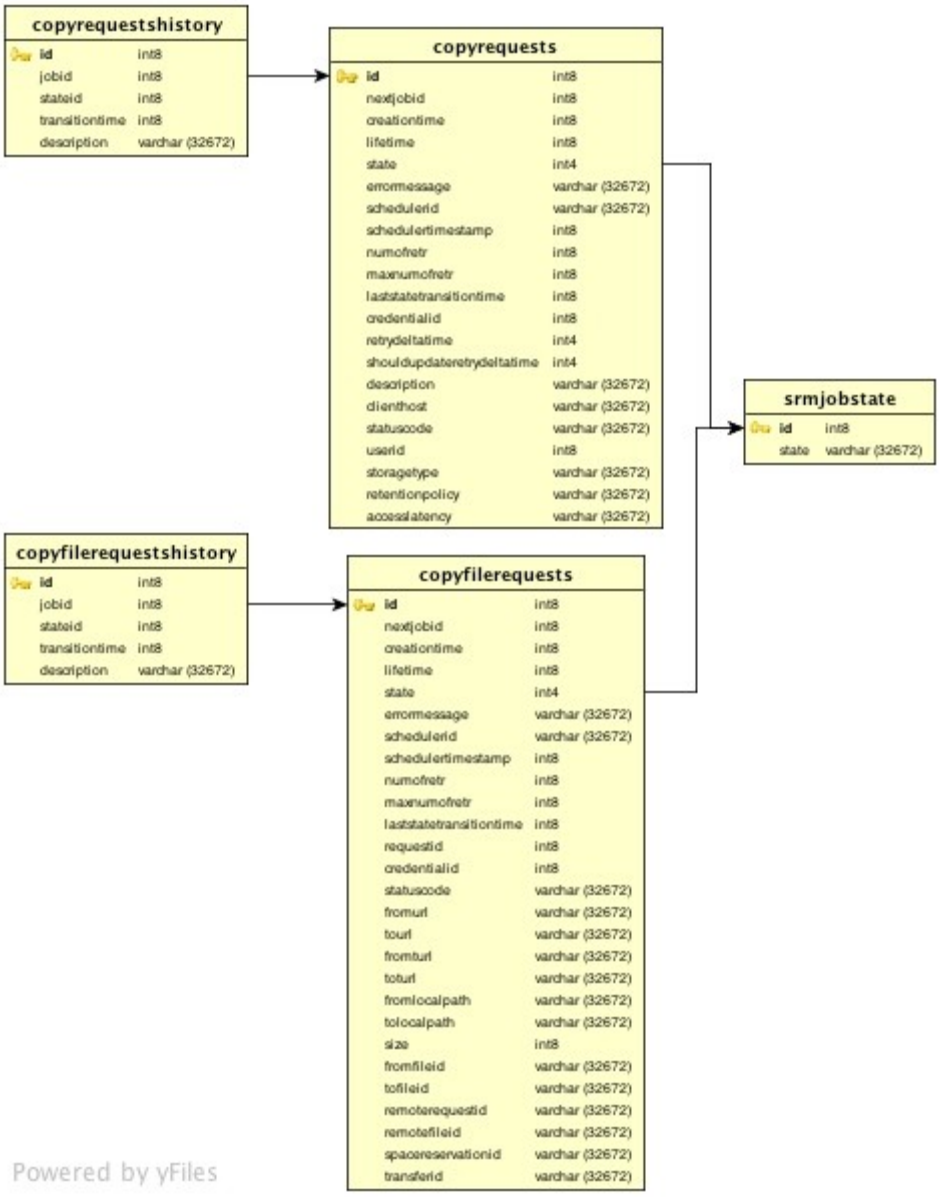
Postgres specific implementations of these interfaces, using JDBC technology, are located in package org.dcache.srm.request.sql.

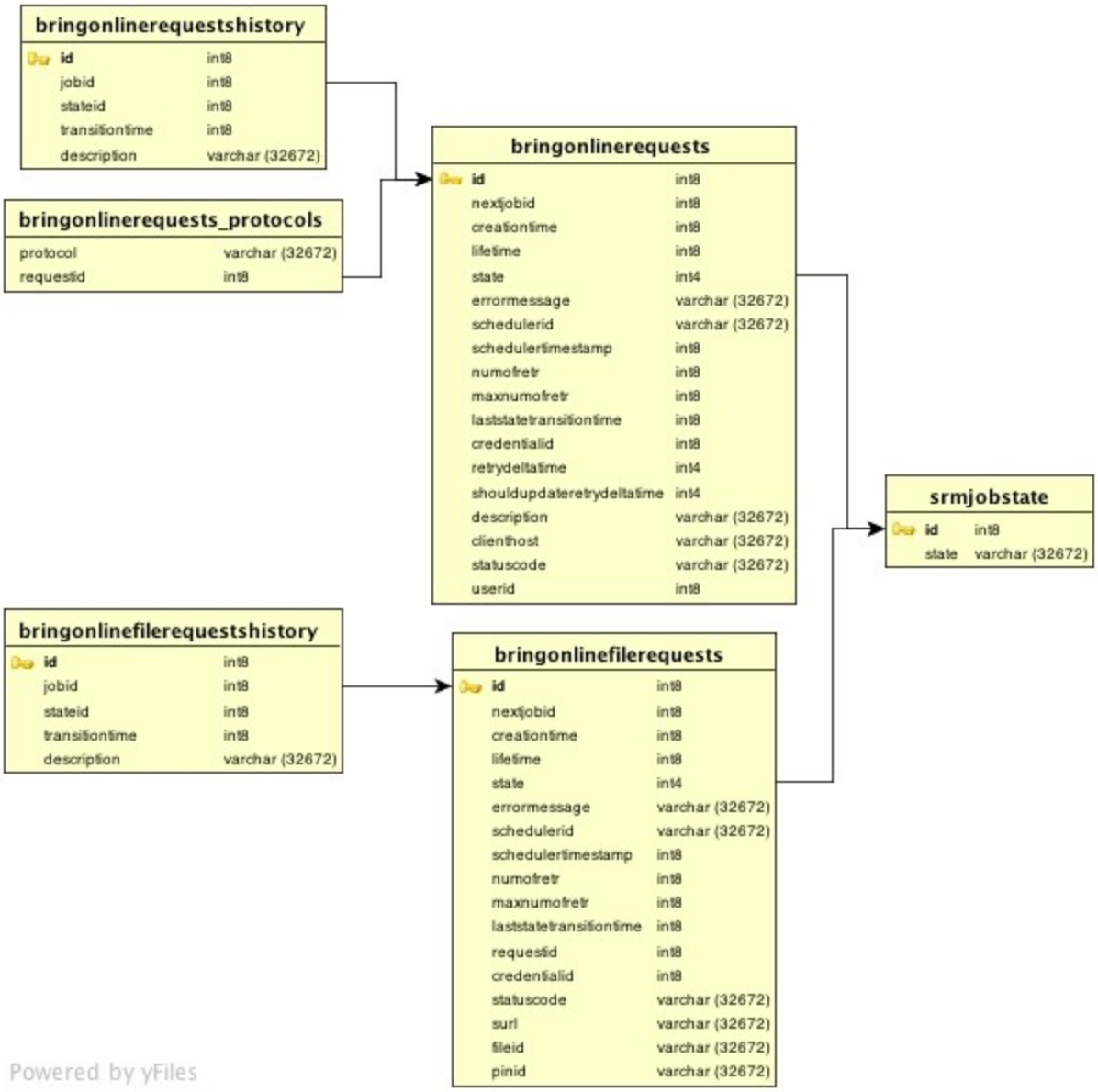
You can have a look at a specific example of the SrmWatch monitoring interface which is using there records about the state stored in Database at [12].

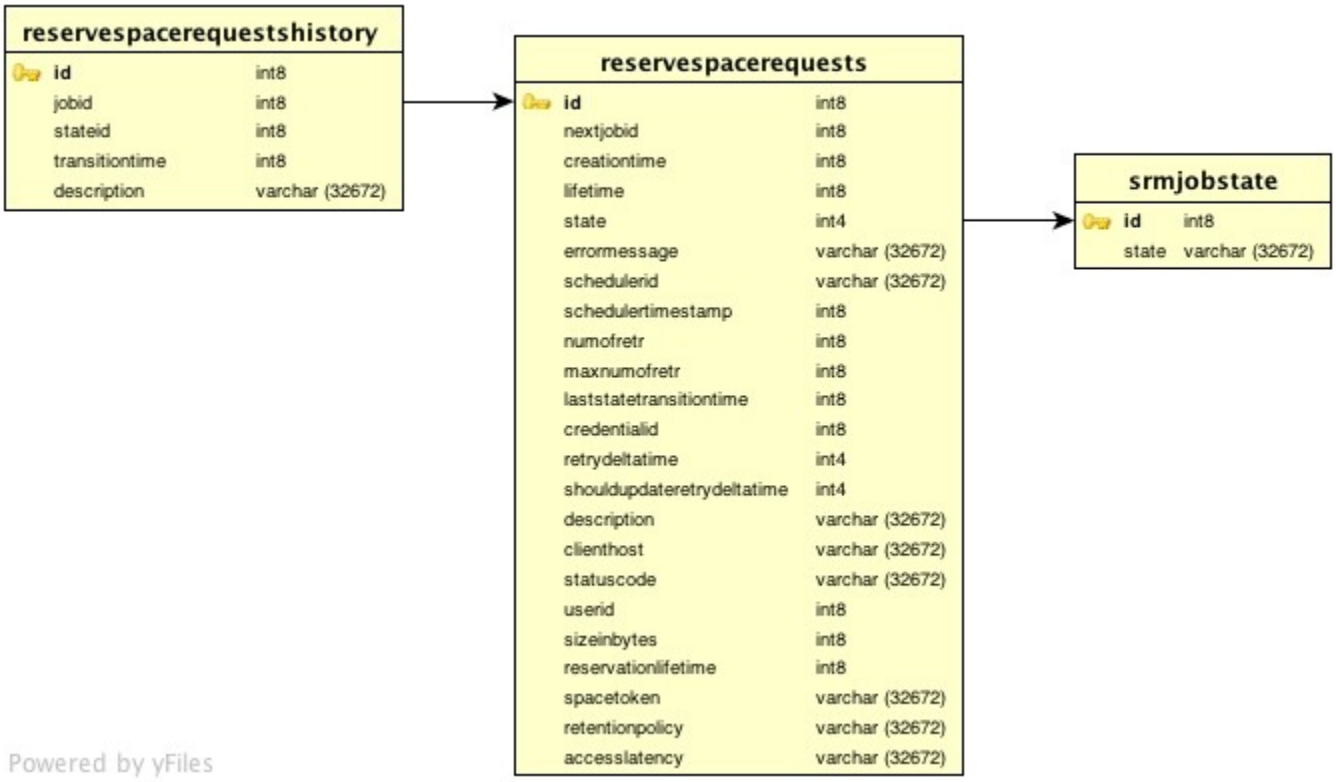
Bellow are the ER Diagrams for SRM Get, Put, Copy, Bring Online and Reserve Space requests and file requests:




putrequestshistory
id int8
jobid int8
stateid int8
transitiontime int8
description varchar (32672)
putrequests_protocols
protocol varchar (32672)
requestid int8
putrequests
id int8
nextjobid int8
creationtime int8
lifetime int8
state int4
errormessage varchar (32672)
schedulerid varchar (32672)
schedulertimestamp int8
numofretr int8
maxnumofretr int8
laststatetransitiontime int8
credentialid int8
retrydeltatime int4
shouldupdateretrydeltatime int4
description varchar (32672)
clienthost varchar (32672)
statuscode varchar (32672)
userid int8
srmjobstate
id int8
state varchar (32672)
putfilerequestshistory
id int8
jobid int8
stateid int8
transitiontime int8
description varchar (32672)
putfilerequests
id int8
nextjobid int8
creationtime int8
lifetime int8
state int4
errormessage varchar (32672)
schedulerid varchar (32672)
schedulertimestamp int8
numofretr int8
maxnumofretr int8
laststatetransitiontime int8
requestid int8
credentialid int8
statuscode varchar (32672)
surl varchar (32672)
turl varchar (32672)
fileid varchar (32672)
parentfileid varchar (32672)
spacereservationid varchar (32672)
size int8
retentionpolicy varchar (32672)
accesslatency varchar (32672)
Powered by yFiles


copyrequestshistory
id int8
jobid int8
stateid int8
transitiontime int8
description varchar (32672)
copyrequests
id int8
nextjobid int8
creationtime int8
lifetime int8
state int4
errormessage varchar (32672)
schedulerid varchar (32672)
schedulertimestamp int8
numofretr int8
maxnumofretr int8
laststatetransitiontime int8
credentialid int8
retrydeltatime int4
shouldupdateretrydeltatime int4
description varchar (32672)
clienthost varchar (32672)
statuscode varchar (32672)
userid int8
storagetype varchar (32672)
retentionpolicy varchar (32672)
accesslatency varchar (32672)
srmjobstate
id int8
state varchar (32672)
copyfilerequestshistory
id int8
jobid int8
stateid int8
transitiontime int8
description varchar (32672)
copyfilerequests
id int8
nextjobid int8
creationtime int8
lifetime int8
state int4
errormessage varchar (32672)
schedulerid varchar (32672)
schedulertimestamp int8
numofretr int8
maxnumofretr int8
laststatetransitiontime int8
requestid int8
credentialid int8
statuscode varchar (32672)
fromurl varchar (32672)
tourl varchar (32672)
fromturl varchar (32672)
toturl varchar (32672)
fromlocalpath varchar (32672)
tolocalpath varchar (32672)
size int8
fromfileid varchar (32672)
tofileid varchar (32672)
remoterequestid varchar (32672)
remotefileid varchar (32672)
spacereservationid varchar (32672)
transferid varchar (32672)
Powered by yFiles


bringonlinerequestshistory
id int8
jobid int8
stateid int8
transitiontime int8
description varchar (32672)
bringonlinerequests_protocols
protocol varchar (32672)
requestid int8
bringonlinerequests
id int8
nextjobid int8
creationtime int8
lifetime int8
state int4
errormessage varchar (32672)
schedulerid varchar (32672)
schedulertimestamp int8
numofretr int8
maxnumofretr int8
laststatetransitiontime int8
credentialid int8
retrydeltatime int4
shouldupdateretrydeltatime int4
description varchar (32672)
clienthost varchar (32672)
statuscode varchar (32672)
userid int8
srmjobstate
id int8
state varchar (32672)
bringonlinefilerequestshistory
id int8
jobid int8
stateid int8
transitiontime int8
description varchar (32672)
bringonlinefilerequests
id int8
nextjobid int8
creationtime int8
lifetime int8
state int4
errormessage varchar (32672)
schedulerid varchar (32672)
schedulertimestamp int8
numofretr int8
maxnumofretr int8
laststatetransitiontime int8
requestid int8
credentialid int8
statuscode varchar (32672)
surl varchar (32672)
fileid varchar (32672)
pinid varchar (32672)
Powered by yFiles


reservespacerequestshistory
id int8
jobid int8
stateid int8
transitiontime int8
description varchar (32672)
reservespacerequests
id int8
nextjobid int8
creationtime int8
lifetime int8
state int4
errormessage varchar (32672)
schedulerid varchar (32672)
schedulertimestamp int8
numofretr int8
maxnumofretr int8
laststatetransitiontime int8
credentialid int8
retrydeltatime int4
shouldupdateretrydeltatime int4
description varchar (32672)
clienthost varchar (32672)
statuscode varchar (32672)
userid int8
sizeinbytes int8
reservationlifetime int8
spacetoken varchar (32672)
retentionpolicy varchar (32672)
accesslatency varchar (32672)
srmjobstate
id int8
state varchar (32672)
Powered by yFiles

Implementation of the main SRM Operations

Common properties of the SRM Requests and File Requests

Srm Requests and File Requests, as it was explained above, are the objects created on the SRM Server side that are used for storage of the state and executing of the non-blocking SRM operations that, form the client point of view, require a sequence of the WS level operations to complete. In dCache they have following commend properties and behavior.

Each request have Unique Long ID, which is used as a v2.2 “token” to identify the request. It also has a state. It has a lifetime. If the request does not reach the final state “lifetime” millisecond after its creation, the request is “canceled”. A request has [a reference to] the credential and the authorization objects. Requests that contain File Requests have an array of FileRequests.

SrmPrepareToGet

SrmGetRequest and its SrmGetFileRequests is created as a result of the SrmPrepareToGet (get in v1.1) WS operation, its status is queried using SrmStatusOfGetRequest (getRequestStatus in v1.1), it can be terminated after successful transfers using srmRelease (setStatus(“Done”) in v1.1) , and it can be aborted by srmAbortRequest or srmAbortFiles.

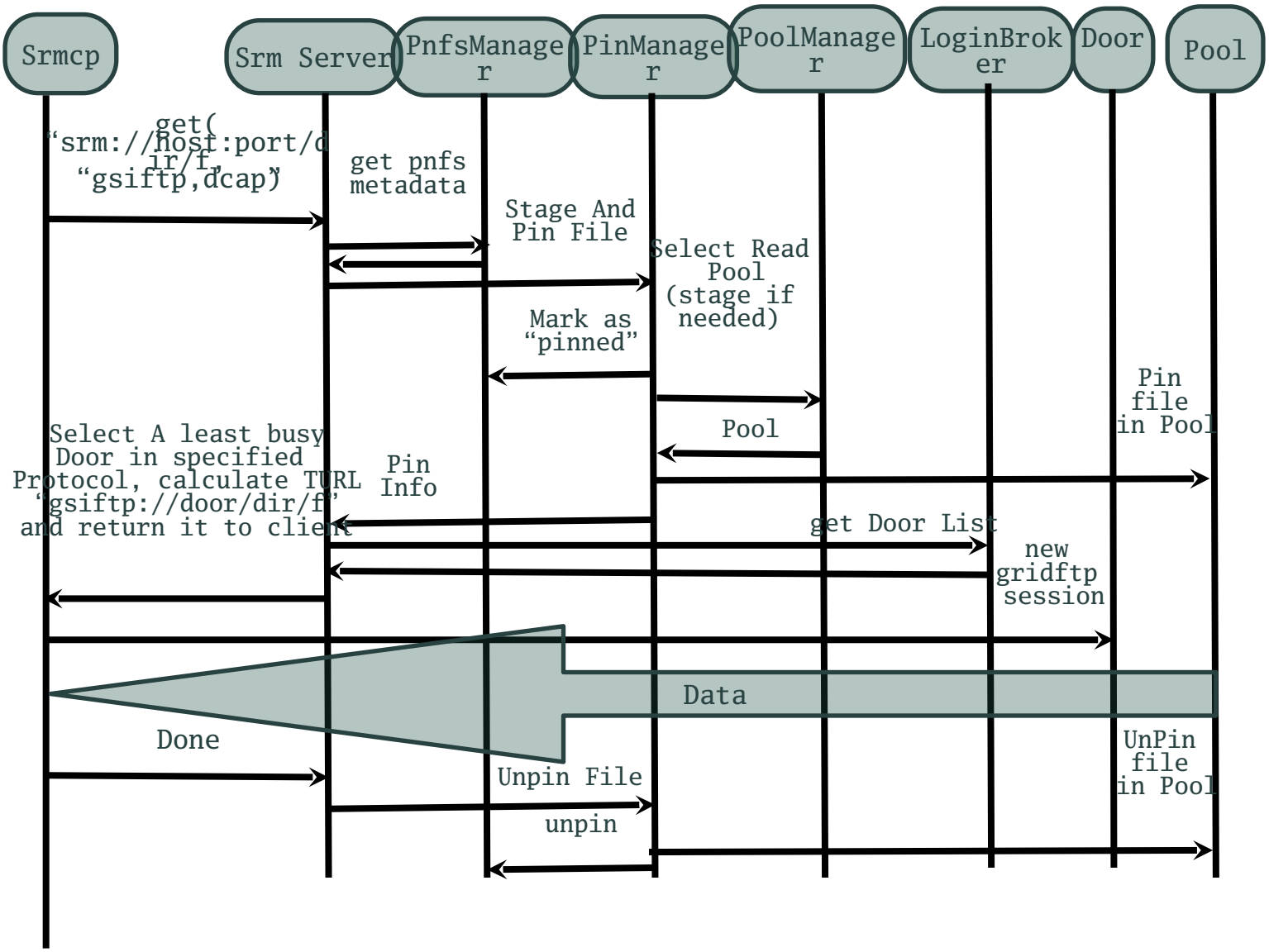
SrmGetRequest is a container of SrmGetFileRequests, there is no preliminary actions that need to be performed on behalf of all the files in the requests, so the SrmGetRequest is not scheduled for execution, instead each of its SrmGetFileRequests is.

Diagram on Picture 5. provides a high level view of what dCache operations are performed by the SrmGetFileRequest in order to satisfy client's srmPrepareToGetRequest.

The list of the AbstractStorageElement methods and dCache Messages used is :

#	SRM AbstractStorageElement view	dCache view
1	getFileInfo	PnfsGetStorageInfoMessage

#	SRM AbstractStorageElement view	dCache view
2	canRead	AuthorizationRecord.UserCanRead || AuthorizationRecord.GroupCanRead || AuthorizationRecord.WorldCanRead
3	pinFile	PinManagerPinMessage
4	getGetTurl	relies on periodic LoginBrokerInfo retrieval by “ls -binary” command.
5	unpinFile	PinManagerUnpinMessage
6		
7		
8		
9		


Srmcp
Srm Server
PnfsManager
PinManager
PoolManager
LoginBroker
Door
Pool
get("srm://host:port/dir/f", "gsiftp,dcap")
get pnfs metadata
Stage And Pin File
Select Read Pool (stage if needed)
Mark as "pinned"
Pin file in Pool
Pool
Select A least busy Door in specified Protocol, calculate TURL "gsiftp://door/dir/f" and return it to client
Pin Info
get Door List
new gridftp session
Data
Done
UnPin file in Pool
Unpin File
unpin

Picture 5.

SrmPrepareToPut

SrmPutRequest and its SrmPutFileRequests is created as a result of the SrmPrepareToPut (put in v1.1) WS operation, its status is queried using SrmStatusOfPutRequest (getRequestStatus in v1.1), it can be terminated after successful transfers using srmPutDone (setStatus("Done") in v1.1) , and it can be aborted by srmAbortRequest or srmAbortFiles.

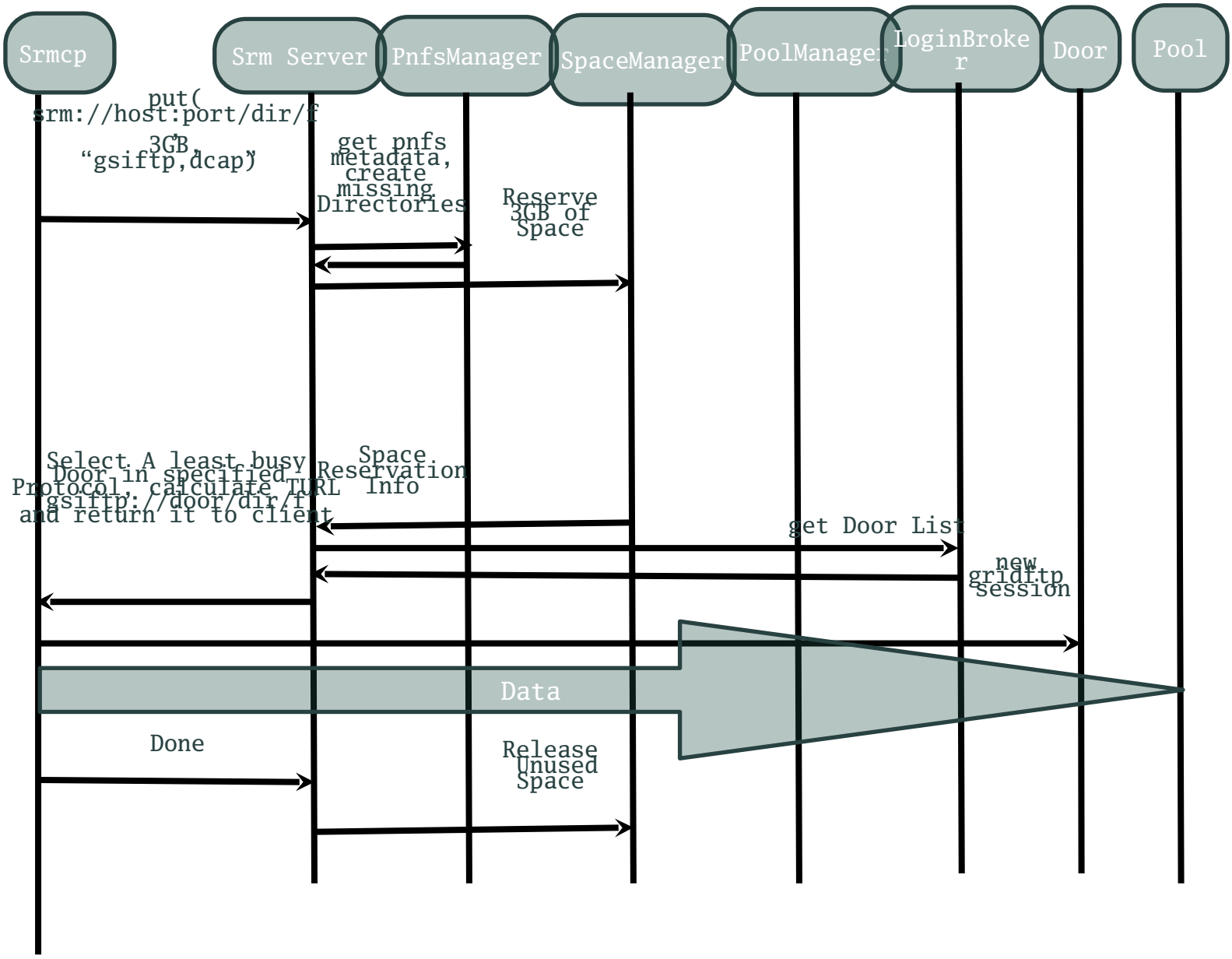
SrmPutRequest is a container of SrmPutFileRequests, and again there is no preliminary actions that need to be performed on behalf of all the files in the requests, so the SrmPutRequest is not scheduled for execution, instead each of its SrmPutFileRequests is.

Diagram on Picture 6. provides a high level view of what dCache operations are performed by the SrmPutFileRequest in order to satisfy client's srmPrepareToPutRequest.

The list of the AbstractStorageElement methods and dCache Methods or Messages used is :

#	SRM AbstractStorageElement view	dCache view
1	PrepareToPutFile	PnfsGetStorageInfoMessage (file) PnfsGetFileMetaDataMessage (parent(file) if not found: PnfsGetFileMetaDataMessage (parent(parent)) if not found: .... PnfsGetFileMetaDataMessage (parent(....(parent(file))) PnfsCreateDirectoryMessage(parent(parent(....)) ... PnfsCreateDirectoryMessage(parent(file))
2	if no space token is given and implicit space reservation enabled: srmReserveSpace	diskCacheV111.services.space.message.Reserve
3	If space token is present srmMarkSpaceAsBeingUsed	diskCacheV111.services.space.message.Use
4	getPutTurl	relies on periodic LoginBrokerInfo retrieval by “ls -binary” command.
5	If we marked space for use srmUnmarkSpaceAsBeingUsed	diskCacheV111.services.space.message.CancelUse

#	SRM AbstractStorageElement view	dCache view
6	If we reserved space: srmReleaseSpace	diskCacheV111.services.space.message.Release
7		
8		
9		


Srmcp
Srm Server
PnfsManager
SpaceManager
PoolManager
LoginBroke
r
Door
Pool
put(
srm://host:port/dir/f
3GB,
“gsiftp,dcap”)
get pnfs
metadata,
create
missing
Directories
Reserve
3GB of
Space
Select A least busy
Door in specified
Protocol, calculate TURL
“gsiftp://door/dir/f”
and return it to client
Space
Reservation
Info
get Door List
new
gridftp
session
Data
Done
Release
Unused
Space

Picture 6.

SRMCopy

SrmCopyRequest and its SrmCopyFileRequests is created as a result of the SrmCopy (get in v1.1) WS operation, its status is queried using SrmStatusOfCopyRequest (getRequestStatus in v1.1), it gets into the final state automatically upon completion of the transfers without waiting for specific commands from the client , and it can be aborted at any time by srmAbortRequest or srmAbortFiles.

SRMCopy is different from the srmPrepareToPut and srmPrepareToGet in the following ways. It does not only prepare the storage system for the transfer of the data in or out of it, it actually performs the transfer. In order to accomplish it, the system needs to play the role of the srm client when talking to the second srm system, that is the source or the destination of the data. Depending on the direction of the transfer the srm copy performs different sequence of actions, so it makes sense to consider Copy in the pull an push mode separately.

SRMCopy in pull mode

CopyRequest in the pull mode negotiates the transfer urls with the remote system using srm prepareToGet function. It does it for all files in the request at once on the level of the Request. The classes that implements the logic of the client are org.dcache.srm.client.RemoteTurlGetterV1 and org.dcache.srm.client.RemoteTurlGetterV2. As soon as a TURL for a given file becomes available, individual CopyFileRequest is scheduled.

CopyFileRequest performs all the same actions that srmPrepareToPut does, with the exception of calculation of TURL and giving it to the client. Instead it requests the Storage System to perform the transfer itself, which is than scheduled and executed by “RemoteGsiftpTransferManager” service in dCache.

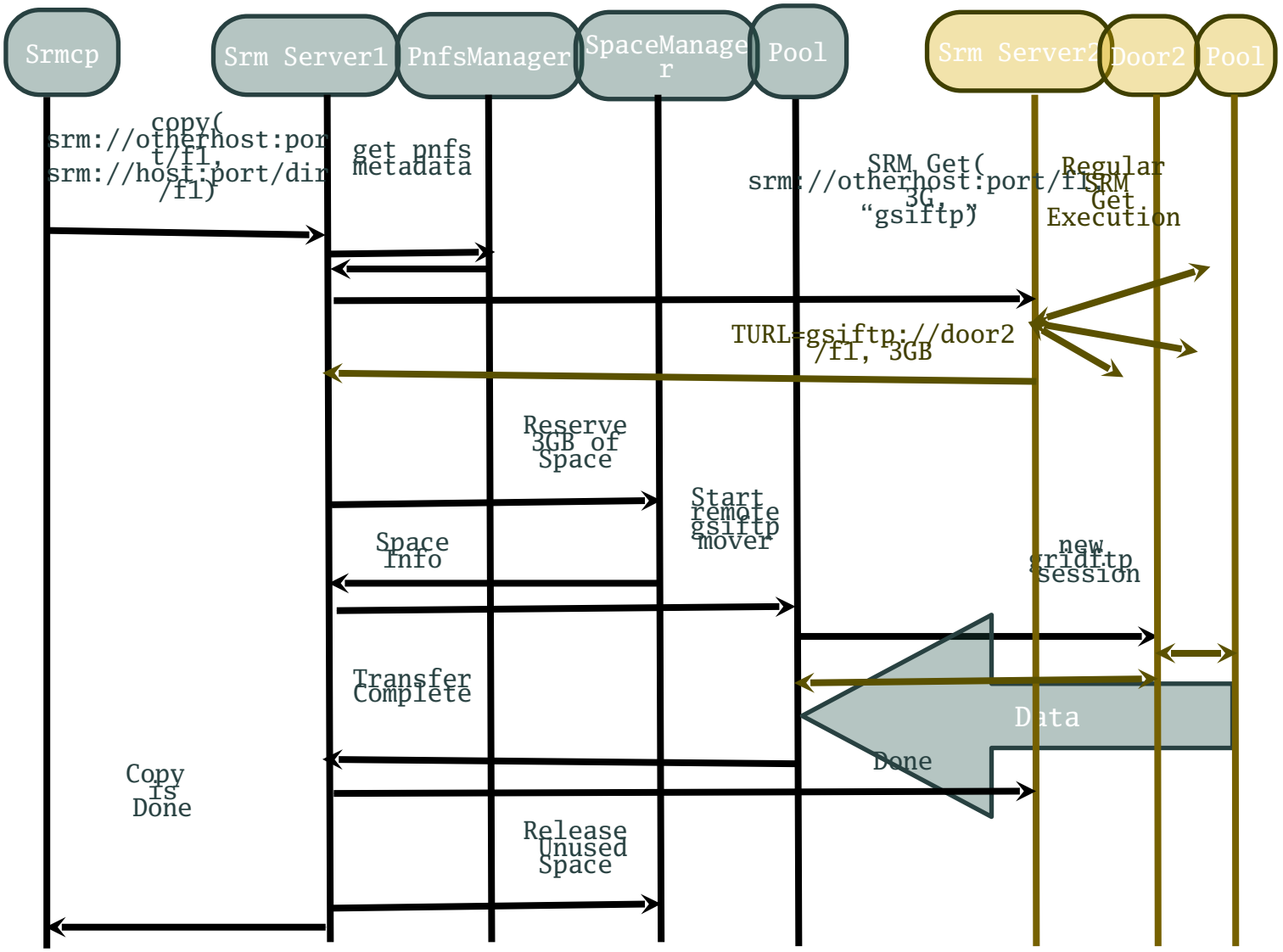
Once the pool starts the mover that performs the transfer, SRM delegates client's credentials to the mover, so that the pool can authorize itself on behalf of the user, with the gridftp server serving the source file.

Diagram on Picture 7. provides a high level view of what dCache operations are performed by the SrmCopyRequest and SrmCopyFileRequests in order to satisfy client's srmCopyRequest in pull mode.

The list of the AbstractStorageElement methods and dCache Methods or Messages used is :

#	SRM AbstractStorageElement view	dCache view
1	getFileMetaData	PnfsGetStorageInfoMessage (file)
2	RemoteTurlGetterV2.run()	
1	PrepareToPutFile	PnfsGetStorageInfoMessage (file) PnfsGetFileMetaDataMessage (parent(file) if not found: PnfsGetFileMetaDataMessage (parent(parent)) if not found: .... PnfsGetFileMetaDataMessage (parent(.... (parent(file))) PnfsCreateDirectoryMessage(parent(parent(. ...)) ... PnfsCreateDirectoryMessage(parent(file))
2	if no space token is given and implicit space reservation enabled: srmReserveSpace	diskCacheV111.services.space.message.Reserve
3	If space token is present srmMarkSpaceAsBeingUsed	diskCacheV111.services.space.message.Use
4	getFromRemoteTURL	diskCacheV111.vehicles.transferManager.RemoteGsiftpTransferManagerMessage

#	SRM AbstractStorageElement view	dCache view
5	If we marked space for use srmUnmarkSpaceAsBeingUsed	diskCacheV111.services.space.message.CancelUse
6	If we reserved space: srmReleaseSpace	diskCacheV111.services.space.message.Release


Srmcp
Srm Server1
PnfsManager
SpaceManager
Pool
Srm Server2
Door2
Pool
copy(
srm://otherhost:port/f1,
srm://host:port/dir/f1)
get pnfs metadata
SRM Get(
srm://otherhost:port/f1,
3G,
"gsiftp")
Regular SRM Get Execution
TURL=gsiftp://door2/f1, 3GB
Reserve 3GB of Space
Start remote gsiftp mover
Space Info
new gridftp session
Transfer Complete
Data
Done
Copy is Done
Release Unused Space

Picture 7.

SRMCopy in push mode

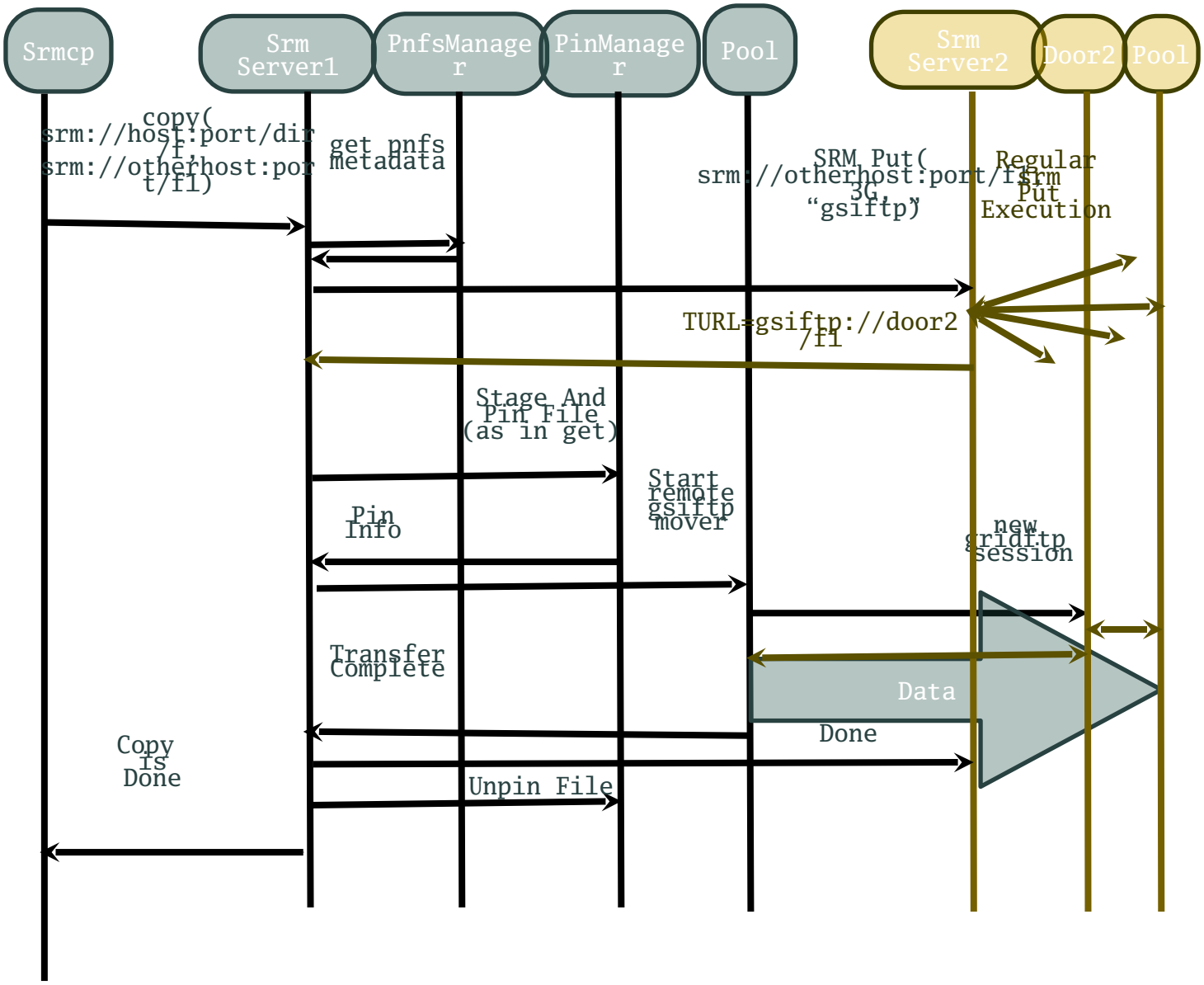
CopyRequest in the push mode negotiates the transfer urls with the remote system using srm prepareToPut function. It does it for all files in the request at once on the level of the Request. The classes that implements the logic of the client are org.dcache.srm.client.RemoteTurlPutterV1 and org.dcache.srm.client.RemoteTurlPutterV2. As soon as a TURL for a given file becomes available, individual CopyFileRequest is scheduled.

CopyFileRequest delegates most of the work related to performing the transfer to underlying storage system, which schedules and executes the transfer in "RemoteGsiftpTransferManager" service in dCache. "RemoteGsiftpTransferManager" in its turn perform many actions similar to the ones performed by srmPretareToGet.

Diagram on Picture 8. provides a high level view of what dCache operations are performed by the SrmCopyRequest and SrmCopyFileRequests in order to satisfy client's srmCopyRequest in push mode.

The list of the AbstractStorageElement methods and dCache Methods or Messages used is :

#	SRM AbstractStorageElement view	dCache view
1		
1	RemoteTurlPutterV2.run()	
3	putToRemoteTURL	RemoteGsiftpTransferManagerMessage


Srmcp
Srm Server1
PnfsManager
PinManager
Pool
Srm Server2
Door2
Pool
copy(srm://host:port/dir/f1, srm://otherhost:port/f1)
get pnfs metadata
SRM Put(srm://otherhost:port/f1, 3G, "gsiftp")
Regular srm Put Execution
TURL=gsiftp://door2/f1
Stage And Pin File (as in get)
Start remote gsiftp mover
Pin Info
new gridftp session
Transfer Complete
Data
Done
Copy is Done
Unpin File

Picture 8.

SrmBringOnline

SrmBringOnlineRequest and its SrmBringOnlineFileRequests is created as a result of the SrmBringOnline WS operation, its status is queried using SrmStatusOfBringOnlineRequest, it can be terminated after successful transfers using srmRelease , and it can be aborted by srmAbortRequest or srmAbortFiles.

SrmBringOnlineRequests and SrmBringOnlineFileRequests perform a subset of actions that SrmGet[File]Requests do. SrmBringOnlineFileRequest checks permissions and it pings the files, but it does not calculate and return a TURL and it does not wait for the transfer to be performed by the client and it does not release the pins after the transfer.

SrmReserveSpace

SrmReserveSpaceRequest is created as a result of srmReserveSpace WS level operation, its status is queried using SrmStatusOfReserveSpaceRequest.

This is the Request object that does not contain a set of the corresponding file request. Its function is to reserve a space of specified size and with specified parameters.

#	SRM AbstractStorageElement view	dCache view
1	srmReserveSpace	diskCacheV111.services.space.message.Reserve

SRMLs

SrmLs could be implemented as a scheduled non-blocking operation, as it supports using tokens for future references to the previously submitted ls request, but it is implemented as a one step non-scheduled blocking operation in dCache SRM. SrmLs performs a recursive listing of files and directories in the underlying storage namespace. It returns back a brief or a detailed list of the namespace entries accompanied by their metadata.

The depth of recursion and the level of details are controlled by the input parameters.

Bellow is the list of the AbstractStorageElement methods and dCache Methods or Messages used is

#	SRM AbstractStorageElement view	dCache view
1	getFileMetaData	PnfsGetStorageInfoMessage (file)
2	listDirectoryFiles	Localy mounted pnfs ls,isDir, etc.

SrmRm

SrmRm is a blocking non scheduled operation. It performs deletions of the of files in the underlying storage namespace.

Bellow is the list of the AbstractStorageElement methods and dCache Methods or Messages used is

#	SRM AbstractStorageElement view	dCache view
1	removeFile	PnfsGetStorageInfoMessage (file) PnfsDeleteEntryMessage(file)

dCache specific implementation of the SRM to Storage Interfaces.

Authorization

Implementation of the SRMAuthorization is diskCacheV111.srm.dcache.DCacheAuthorization. In most widely used configuration DcacheAuthorization used gPlazma service to obtain authorization record. The message sent to gPlazma is diskCacheV111.vehicles.X509Info and the result sent back is diskCacheV111.vehicles.AuthenticationMessage. ultimately the result of the authorization is the org.dcache.auth.AuthorizationRecord which implements SRMUser interface.

AbstractStorageInterface

Implementation of the AbstractStorageInterface is diskCacheV111.srm.dcache.Storage, which is also a cell in dCache application. In addition of providing dCache specific implementations to all of the AbstractStorageInterface methods, it also contains implementations of many of the admin functions exposed via dCache admin interface, such as “ls” for listing of the pending srm requests, “info” for providing status information, “cancel” for termination of a specific request.

diskCacheV111.srm.dcache.Storage relies on a large number “companion” classes for communication with the other service of the dCache application. The companions are implementation of the CellMessageAnswerable interface, which allows to send the message to other service and receive the notification about the reply asynchronously, via invocation of the CellMessageAnswerable.answerArrived method. Almost all of the SRM Cell to dCache communication is build around this mechanism. Only methods that are called from the synchronous SRM functions are implemented using synchronous cell message exchanges via sendAndWait() calls. The implementation of the specific AbstractStorageInterface method ofter requires a series of message exchanges. In this case companion behaves like a state machine, that changes state with each message exchange stage accomplished until it reaches the final state, when it can notify SRM about the success of failure of the specific operation, using the Callback interface passed as an argument to the Storage method. This mechanism allows SRM code not to block a thread while waiting other dCache services to reply, instead SRM returns the thread to the thread pool and retrieves the thread only when more processing of the request on the srm side is required. Please note that he state machines themselves are driven by a thread pool separate from those used by the scheduler described on page 15 and that this thread pool is shared by all implementations. Therefore the code inside these state machines is minimal and mostly equivalent to sending a next message to a remote cell or rescheduling the SRM request for execution and does not perform any blocking operations itself.

The companions used by diskCacheV111.srm.dcache.Storage are

AdvisoryDeleteCompanion

GetFileInfoCompanion

PinCompanion

PutCompanion

PutInSpaceCompanion

ReleaseSpaceCompanion

RemoveFileCompanion

ReserveSpaceCompanion

SrmMarkSpaceAsBeingUsedCompanion

SrmReleaseSpaceCompanion

SrmReserveSpaceCompanion

SrmUnmarkSpaceAsBeingUsedCompanion

UnpinCompanion

Other dCache Services developed specifically to support SRM.

Gsiftp Transfer Manager, Http Transfer Manager and Copy Manager

Transfer Managers accept requests for performing a Local or Remote File transfer into a Local or Remote file.

Gsiftp can transfer to or from a remote gridftp server, Http Transfer Manager can only transfer Files from remote Http servers into dCache. And Copy manager allows to copy file internally from one local path into another (it uses dcap data transfer protocol for pool to pool communication

All transfer managers have a similar logic.

When transfer request is for copy into a local file arrives, Transfer manager does the following

1. Get parent Directory metadata using PnfsGetFileMetaDataMessage, check permissions

2. Create a namespace entry using PnfsCreateEntryMessage
3. Get file metadata PnfsGetStorageInfoMessage
4. Select pool using PoolMgrSelectWritePoolMsg
5. Start the mover on the pool using PoolAcceptFileMessage
6. once the pool replies with the Transferfinished, report the result by sending reply to SRM.

When request is for copy from dCache into a remote server, the logic is a bit simpler

1. Get file metadata PnfsGetStorageInfoMessage, check permissions
2. Select pool using PoolMgrSelectReadPoolMsg
3. Start the mover on the pool using PoolDeliverFileMessage
4. once the pool replies with the Transferfinished, report the result by sending reply to SRM.

In reality there are more messages sent around to support premature interruption of the transfers. Also in case of GruidftpTransferManager and separate step for delegation of the credentials from the SRM to the pool is needed.

In all cases the mover on the pool is playing a role of the (Gsiftp, http or dcap) client to a remote system and it has itself a rather complex logic.

The code for the transfer manager cells can be found in

diskCacheV111.services.TransferManager

diskCacheV111.services.GsiftpTransferManager

diskCacheV111.doors.RemoteHttpTransferManager

diskCacheV111.doors.CopyManager

the code for the movers is in

diskCacheV111.movers.RemoteGsiftpTransferProtocol_1

diskCacheV111.movers.RemoteHttpTransferProtocol_1

diskCacheV111.movers.DCapClientProtocol_1

Pin Manager

PinManager supports the following commands (messages):

PinManagerPinMessage – this creates the new pin on the file, and pins the file in the pool if it is not pinned. It it is pinned by a different pin, it just adds the pin record and possibly extends the pin in the pool.

PinManagerUnpinMessage – If the file has other current pins, the pin record will be removed, while the file will remain pinned in the pool. If the file is not pinned anymore by any other pin, the File is unpinned in the pool too.

PinManagerExtendLifetimeMessage – This operation results in the update of the lifetime of the pin in the database and possibly on the pool.

PoolRemoveFilesMessage -This notification comes from the PnfsManager if the file was removed from the namespace. In case of receipt of this message all records for file are removed.

The table bellow summarized the PinManager to other dCache communication.

	PinManager operation	dCache view
	PinManagerPinMessage	PoolMgrSelectReadPoolMsg (pnfsid, params) PoolSetStickyMessage(on, pnfsid) or just PoolSetStickyMessage(pnfsid,new lifetime)
	PinManagerUnpinMessage	PoolSetStickyMessage(off, pnfsid)
	PinManagerExtendLifetimeMessage	PoolSetStickyMessage(pnfsid,new lifetime)

Space Manager

Somewhat outdated space manager design document can be found at 54. the code is in

diskCacheV111.services.space java package, the messages are in
diskCacheV111.services.space.messages

Appendix A. Dcache SRM Code structure

SRM Code Structure

There are three cvs modules in desy repository containing srm related code:

1. srm module contains pure srm code not related to any particular underlying storage and unixfs/enstore implementation.
2. srmclient – client software -please see srm/doc/SrmClientCodeStructure.sxw for details.
3. Sandbox contains dcache and dcache specific srm code.

The dCache SRM code tree

```
:-----diskCacheV111           |
: :-----srm                   |
: : :-----server              | Legacy SRM V1.1 WS server code
:-----org                     |
: :-----dcache                |
: : :-----srm                 | Main SRM Interfaces
: : : :-----client            | SRM and Gridftp client code
: : : : :-----axis            | Axis specific client code
: : : : :-----axis1           | -||-
: : : :-----handler           | handlers for SRM V2.2 functions
: : : :-----lambdastation     | Lambda Station code
: : : :-----qos               |
: : : : :-----lambdastation   | Lambda Station QoS plugin
: : : : :-----terapaths       | Terapath QoS plugin
: : : :-----request           | SRM Get/Put/Copy/BOL requests
: : : : :-----sql             | Persistence code for requests
: : : :-----scheduler         | SRM Requests Scheduler code
: : : : :-----policies        | policies for Scheduler
: : : :-----security          | Legacy security code
: : : :-----server            | Axis SRM V1.1 and V2.2 servers
: : : :-----unixfs            | Demo Unix FS implementation
: : : :-----util              | Various utility classes/config
: : : : :-----events          | Events
: : : :-----v2_1              | Axis Auto Generated code for V1.1
: : : :-----v2_2              | Axis Auto Generated code for V2.2
```

Srm module package by package break down

Package diskCacheV111.srm.* - stubs for srm v1.1 Web Services interface generated

with GLUE WS Toolkit from “The Mind Electric”. These can not be moved to different package because it makes web service incompatible with other srm implementations and older clients:
class ISRM.java – srm interface

classes FileMetaData.java, RequestStatus.java, RequestFileStatus.java -parameters and return value types.

Please read http://sdm.lbl.gov/srm-wg/doc/SRM.spec.v2.1.final.pdf for details and meaning of these.

Classes IInformationProvider.java StorageElementInfo.java – used for the implementation of information service provider interface defined in the storage element glue schema,
please see
http://www.cnaf.infn.it/~sergio/datatag/glue/v11/SE/GlueSE_DOC_V_1_1.html
for details.

class **diskCacheV111.srm.server.Server** – implementation of srm v1.1 server
-responsible for network connectivity, authentication and partial authorization and passing parameters to org.dcache.srm.SRM class.

Package org.dcache.srm.* - major srm internal interfaces and classes
Logger.java -logger interface, extended by AbstractStorageElement.java
SRMUser.java – abstraction of the storage specific user class
AbstractStorageElement.java - interface between srm and underlying storage.

The following callback interfaces are passed as arguments in AbstractStorageElement functions. They are used for asynchronous notifications of completion of execution of the functions.
ReserveSpaceCallbacks.java
AdvisoryDeleteCallbacks.java

PinCallbacks.java
UnpinCallbacks.java
PrepareToPutCallbacks.java
CopyCallbacks.java
PrepareToPutInSpaceCallbacks.java
GetFileInfoCallbacks.java
ReleaseSpaceCallbacks.java

Exceptions:
SRMException.java
SRMAuthorizationException.java - ...
BadSRMObjectException.java – added by Leo
SRMAuthorization.java – abstraction of the storage specific authorization mechanism.

Srm class contains one function for each srm v1 and srm v2 specified function, it is responsible for either creating and scheduling srm requests or directly executing srm functions, which do not require scheduling. Execution of these functions will eventually lead to execution of the functions of the AbstractStorageElement implementation.

Package org.dcache.srm.security.*
SslGsiSocketFactory.java – implements socketfactory interface from Glue and is used for gridifying Glue http server and client. (Various grid server and regular sockets are inner classes of SslGsiSocketFactory.java).

Tomcat*java classes are used for gridyfying tomcat server (used for running axis servlet, which is in its turn is used for running srm v2.1 WS (Web Services) server).

Package org.dcache.srm.scheduler.* contains an srm scheduler implementation. Please see SRM.Request.Scheduler.Design.sxw in srm/doc directory for details. SRM.Request.Scheduler.Design.sxw was written before the implementation commenced and does reflect general principals though many details are incorrect.

scheduler/Scheduler.java is the main class that is responsible for scheduling
scheduler/Job.java abstract class representing the job that needs execution
scheduler/State.java instances of this type represent various states jobs can assume

scheduler/ModifiableQueue.java - utility class representing queue
scheduler/JobStorage.java - interface, implementation of which will provide persistent storage for jobs, Scheduler knows how to restore previously saved jobs from the instance of JobStorage upon restart.

scheduler/JobCreator.java interface that represent the owner of the job, job creator has a priority and might have certain number of jobs in the queues or running in the scheduler. These numbers are taken in consideration when next job is selected for the execution.
scheduler/, scheduler/NonFatalJobFailure.java – job execution method (run()) can throw these exceptions. if NonFatalJobFailure is thrown, job execution is retried after a predefined period of time (up to max num. of retries). Otherwise if FatalJobFailure is throws, job is "Failed".

scheduler/JobCreatorStorage.java interface of class that provides persistent storage for Job creators.
scheduler/JobIdGenerator.java - ...

Sample implementations:
scheduler/HashtableJobCreatorStorage.java
scheduler/SimpleIdGenerator.java
scheduler/HashtableJobStorage.java

Internal classes:
scheduler/StateChangedEvent.java
scheduler/JobStorageAddedEvent.java
scheduler/IllegalStateTransition.java

Package org.dcache.srm.request* contains implementations of various srm request types as subclasses of scheduler Job, so that they can be executed by srm scheduler

Package org.dcache.srm.request.sql.* contains persistent storages for these requests.

Package org.dcache.srm.client.* srm client libraries.

Package org.dcache.srm.v2_1 srm v2.1 interface stubs generated using apache axis

wsdl2java tool (done by Leo)

Package org.dcache.srm.server – srm v2.1 server libraries

dCache module code subtree related to SRM

```
:-----diskCacheV111          |
: :-----doors                |  dCache Doors (control channels)
: :-----movers               |  dCache Movers (data channels)
: :-----services             |
: : :-----authorization      |   old gPlazma code
: : : :-----classes          |    -||-
: : : : :-----gplazma        |    -||-
: : : : : :-----gplazmalite  |    -||-
: : : : : :-----vo-mapping   |    -||-
: : : : : : :-----endorsed   |    -||-
: : : : : : :-----etc        |    -||-
: : : : : : :-----lib        |    -||-
: : :-----space              |  SRM Space Manager
: : : :-----message          |  SRM Space Manager messages
: :-----srm                  |
: : :-----dcache             |  dCache implementation of SRM
: :                          |  interfaces
: :-----vehicles             |  dCache communication messages
: : :-----transferManager    |  messages for TransferManagers
:-----org                    |
: :-----dcache               |
: : :-----auth               |  Authorization Records
: : : :-----gplazma          |  gPlazma code
: : : :-----persistence      |  Persistence for Auth Records
: : :-----services           |
: : : :-----pinmanager1      |  SRM Pin Manager
```

Appendix 2. Messages used for SRM - DCache communication

SrmPrepareToGet

1	getFileInfo	PnfsGetStorageInfoMessage
2	canRead	AuthorizationRecord.UserCanRead || AuthorizationRecord.GroupCanRead || AuthorizationRecord.WorldCanRead
3	pinFile	PinManagerPinMessage
4	getGetTurl	relies on periodic LoginBrokerInfo retrieval by "ls -binary" command.
5	unpinFile	PinManagerUnpinMessage

SrmPrepareToPut

1	PrepareToPutFile	... PnfsCreateDirectoryMessage(parent(file))
2	if no space token is given and implicit space reservation enabled: srmReserveSpace	diskCacheV111.services.space.message.Reserve
3	If space token is present srmMarkSpaceAsBeingUsed	diskCacheV111.services.space.message.Use
4	getPutTurl	relies on periodic LoginBrokerInfo retrieval by "ls -binary" command.
	When PutDone called or Request expires: GetFileMetaData	(depends on the SRMUser instance) 3. PoolMgrQueryPoolsMsg + GetCacheLocations if storage info is unknown 4. GetFileSpaceTokens
5	If we marked space for use srmUnmarkSpaceAsBeingUsed	diskCacheV111.services.space.message.CancelUse
6	If we reserved space: srmReleaseSpace	diskCacheV111.services.space.message.Release

SRMCopy (Pull mode)

1	getFileMetaData	PnfsGetStorageInfoMessage (file)
2	RemoteTurlGetterV2.run()	
3	PrepareToPutFile	... PnfsCreateDirectoryMessage(parent(file))
4	if no space token is given and implicit space reservation enabled: srmReserveSpace	diskCacheV111.services.space.message.Reserve
5	If space token is present srmMarkSpaceAsBeingUsed	diskCacheV111.services.space.message.Use
6	getFromRemoteTURL	diskCacheV111.vehicles. transferManager.RemoteGsiftpTransferManagerMessage
7	If we marked space for use srmUnmarkSpaceAsBeingUsed	diskCacheV111.services.space.message.CancelUse
8	If we reserved space: srmReleaseSpace	diskCacheV111.services.space.message.Release

SRMCopy (Pushl mode)

1	RemoteTurlPutterV2.run()	
2	putToRemoteTURL	RemoteGsiftpTransferManagerMessage
3	SRMReserveSpace	
4	srmReserveSpace	diskCacheV111.services.space.message.Reserve

SRMLs

1	getFileMetaData	PnfsGetStorageInfoMessage (file)
2	listDirectoryFiles	Localy mounted pnfs ls,isDir, etc.

SrmRm

1	removeFile	PnfsGetStorageInfoMessage (file) PnfsDeleteEntryMessage(file)

PinManager Messages

1	PinManagerPinMessage	PoolMgrSelectReadPoolMsg (pnfsid, params) PoolSetStickyMessage(on, pnfsid) or just PoolSetStickyMessage(pnfsid,new lifetime)
2	PinManagerUnpinMessage	PoolSetStickyMessage(off, pnfsid)
3	PinManagerExtendLifetimeMessage	PoolSetStickyMessage(pnfsid,new lifetime)

References

[1] Storage Resource Manager Working Group http://sdm.lbl.gov/srm-wg/

[2] SRM V2.2 http://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.070402.html

[3] GSI Overwiew: http://www.globus.org/security/overview.html

[4] http://tomcat.apache.org/

[5] http://ws.apache.org/axis/

[6] http://trac.dcache.org/projects/dcache/wiki/cell

[7] http://www.dcache.org/manuals/cells/docs/index.html

[8] http://dev.globus.org/wiki/CoG_FX_GSI

[9] http://www.dcache.org/

[10] http://trac.dcache.org/projects/dcache/wiki/manuals/workWithdCacheSVN

[11] SRM Request Scheduler design
http://home.fnal.gov/~timur/srm/SRM.Request.Scheduler.Design.pdf

[12] SRMWatch monitoring interface to US CMS T1 dCache SRM system in Fermilab
http://cmsdcam3.fnal.gov:8081/srmwatch/

[13] Results of profiling of US CMS T1 production SRM (dCache 1.9.0-5 ON Mon Jan 26 2009) http://home.fnal.gov/~timur/srm/CMS.SRM.Profiling/

[14] SRM Code Javadocs http://home.fnal.gov/~timur/srm/doc/javadoc/index.html

[15] Srm Space Manager design http://svn.dcache.org/WebSVN/filedetails.php?repname=dCache&path=%2Ftrunk%2Fmodules%2FdCache%2FdiskCacheV111%2Fservices%2FSpaceManagerDesign.doc

[16] The Anatomy of the Grid
http://www.globus.org/alliance/publications/papers/anatomy.pdf

[17] The Physiology of the Grid
http://www.globus.org/alliance/publications/papers/ogsa.pdf